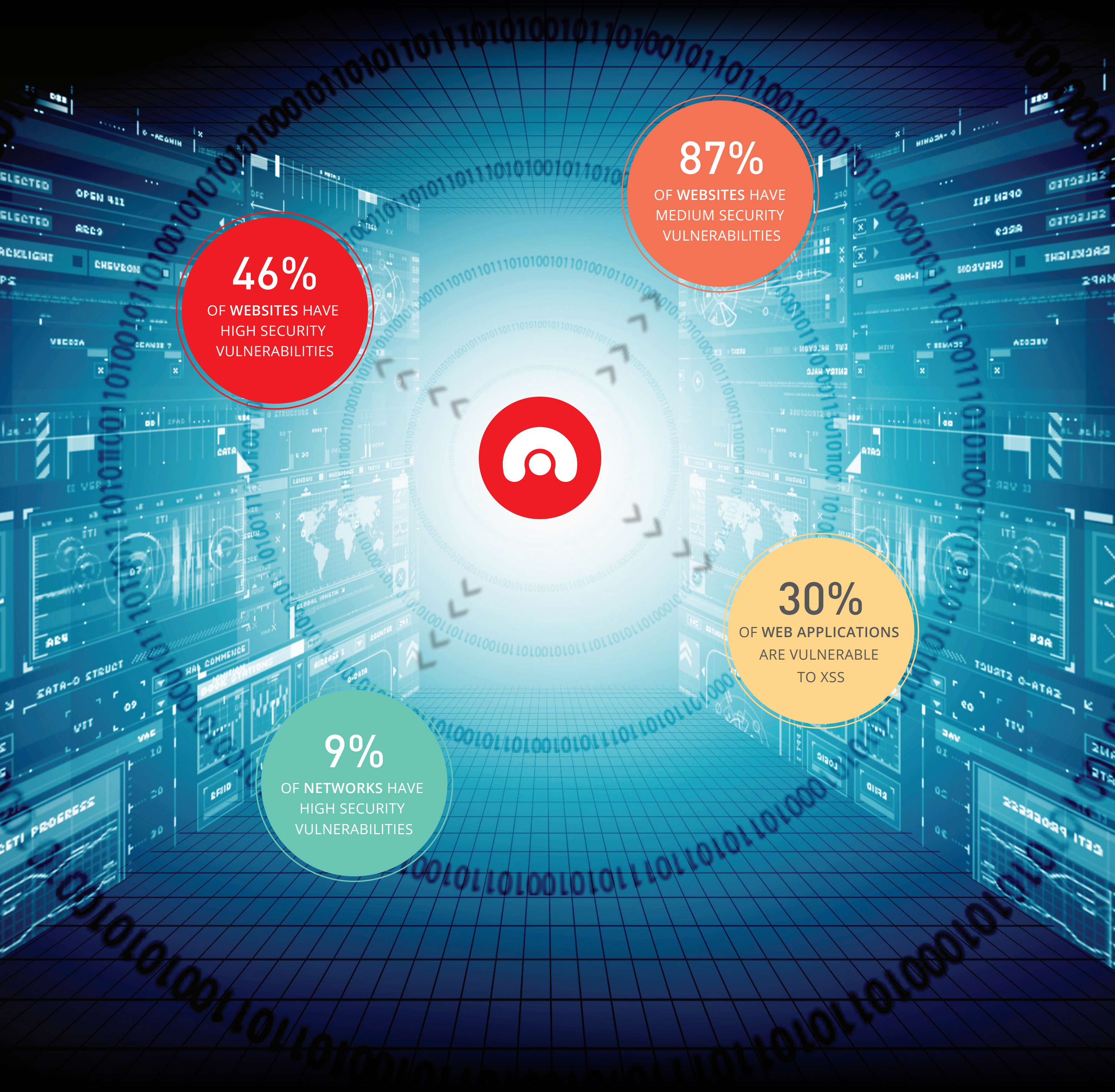
Web Application Vulnerability Report 2019



87% OF **WEBSITES** HAVE MEDIUM SECURITY VULNERABILITIES

46% OF **WEBSITES** HAVE HIGH SECURITY VULNERABILITIES

30% OF **WEB APPLICATIONS** ARE VULNERABLE TO XSS

9% OF **NETWORKS** HAVE HIGH SECURITY VULNERABILITIES

acunetix

Contents

Introduction

Welcome to the 2019 edition of the Acunetix Web Application Vulnerability Report

Every year, Acunetix crunches data compiled from Acunetix Online into a vulnerability testing report that portrays the state of the security of web applications and network perimeters. This year's report contains the results and analysis of vulnerabilities detected over the previous 12 months, across 10,000 scan targets.

Cross-site Scripting (XSS) vulnerabilities, vulnerable JavaScript libraries, and WordPress related issues were found to each claim a significant 30% of the sampled targets. This result continues to reinforce the argument that web applications are both a viable attack vector for attackers and present a low barrier to entry.

In addition, it's becoming clearer that vulnerabilities such as SQL injections – which have been wreaking havoc for years all over the internet – are finally falling in numbers. While this is positive, we also see how other serious vulnerabilities are rising in frequency. Cross-site Scripting (XSS) vulnerabilities claim a major slice of the pie. And it is worrying that a significant amount of targets also include JavaScript libraries with known vulnerabilities in them. These instances constitute a very serious concern for client-side security as a whole.

For the purpose of this analysis, a random sample of 10,000 successfully scanned targets were randomly selected. The analysis focuses predominantly on high and medium severity vulnerabilities found in web applications, as well as perimeter network vulnerability data.

Web applications are steadily becoming more feature rich and complex. This technological complexity is a natural result of increasing consumer demands for an ever more engaging web. In response to this, many Software Development and Operations teams have had to increase the frequency with which they release new versions of their web applications. While DevOps agility provides organizations with faster release cycles, it also means that web security becomes harder to scale.

Web application vulnerabilities are dangerous for many reasons. Public breaches risk damage to a company's brand and reputation. And in an era where privacy is more important than ever, regulations such as GDPR have significantly raised the stakes in terms of financial penalties and data breach disclosure.

The web has changed, and with it, so has the threat landscape. On the client-side, the web browser is now an incredibly powerful ecosystem – one that can be abused to invade victims' privacy, steal their money or even use their CPU to mine cryptocurrency (Cryptojacking). On the server-side, the versatility of the web as a common platform means that, increasingly, web applications and web services have replaced legacy applications. The consequence of this is a more expansive landscape for attackers to exploit. This is particularly true since traditional network-layer security defenses such as firewalls and intrusion detection systems (IDS) are ineffective at detecting or preventing web application attacks.

This report aims to provide you with an overview of the most commonly encountered web application and network perimeter vulnerabilities. Additionally, we hope this report will serve as a signpost for where web vulnerabilities security is headed over the next few years.

Methodology

Data gathered and analyzed for this report was sourced from the results of automated web application and network perimeter security scans run from Acunetix Online, Acunetix's SaaS, cloud-based web and network perimeter scanner. This data was collected over a 12 month period, across 10,000 randomly selected scan targets. Evaluation scans on the intentionally vulnerable Acunetix test web applications were omitted from the scope of this analysis.

The Anatomy of an Automated Web Scan

The majority of automated web application scans run by Acunetix Online are black box or Dynamic Application Security Testing (DAST) scans. This means that the web scanner has no knowledge of the backend code running on the website or the web application it is about to scan.

Acunetix Online can also run gray box or Interactive Application Security Testing (IAST) scans by leveraging Acunetix's AcuSensor technology. This is a sensor that can be installed on the server side for Java, ASP.NET and PHP web applications. AcuSensor brings together the best of dynamic testing, by relaying feedback from sensors within the source code back to Acunetix while it is executing.

Automated web application security testing and vulnerability management processes tend to follow four major stages in most organizations. Each is outlined below.

Crawling and Scanning

Scanning a website or web application first involves crawling it. Acunetix's crawler analyzes the structure of a web application by looking for links and inputs, and by running JavaScript like a real browser. The crawler supports HTML5 and ECMAScript 6 and 7 technologies. It can automatically detect commonly used JSON and XML input schemes, as well as more exotic input schemes like Google Web Toolkit (GWT), in addition to the typical GET and POST parameters.

An accurate crawl is crucial for a scan to have good coverage throughout, since the scanner cannot test a page for vulnerabilities until it is aware of its existence. After completing a crawl, Acunetix automatically tests every page it found for hundreds of security vulnerabilities.

Reporting

After the scan returns some results, the next step is to interpret the findings identified by the scanner. Results are displayed on the Dashboard in real-time, and you can start working on them even before the scan finishes. In addition, scan results can be exported to a comprehensive list of management and compliance reports, including reports for PCI DSS, OWASP Top 10, HIPAA, ISO 27001 and others.

Remediation

Accurate scan results alone are not useful unless the vulnerabilities are fixed. As well as detailed information about discovered vulnerabilities, Acunetix also provides out of the box vulnerability management tools and integrations with issue trackers such as Atlassian, JIRA and GitHub. It can virtually patch vulnerabilities by configuring common Web Application Firewalls. Acunetix easily integrates with Continuous Integration (CI) and Continuous Deployment (CD) tools like Jenkins. Finally, Acunetix even supports Continuous Scanning, running a quick scan every day in addition to a more comprehensive scan once a week to ensure that detected vulnerabilities rapidly move from discovery to remediation.

The Dataset

The data analyzed in this report is gathered from the automated web and network perimeter scans run on the Acunetix Online platform. The analysis in this report is based predominantly on high and medium severity vulnerabilities found in web applications, as well as perimeter network vulnerability data.

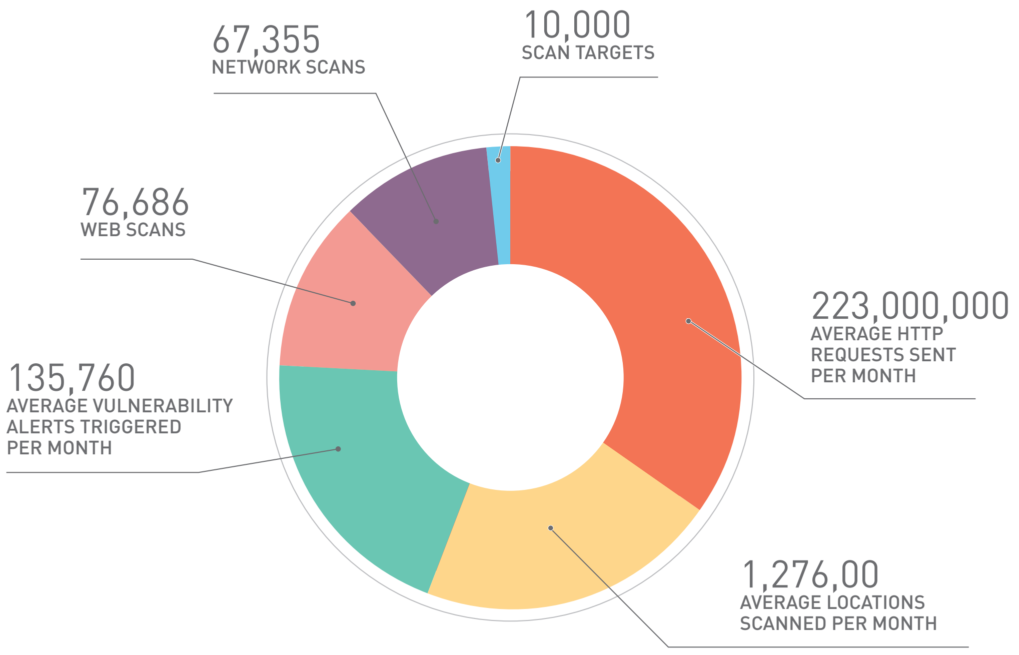


Figure 1.

The Dataset Analysis.

Vulnerabilities at a Glance

This section lists all the detected vulnerabilities from our research.

Vulnerabilities by Type

The charts lists vulnerabilities by type. They are grouped by vulnerability severity level.

High Severity

This chart illustrates the vulnerability types found which fall into our High Severity category.

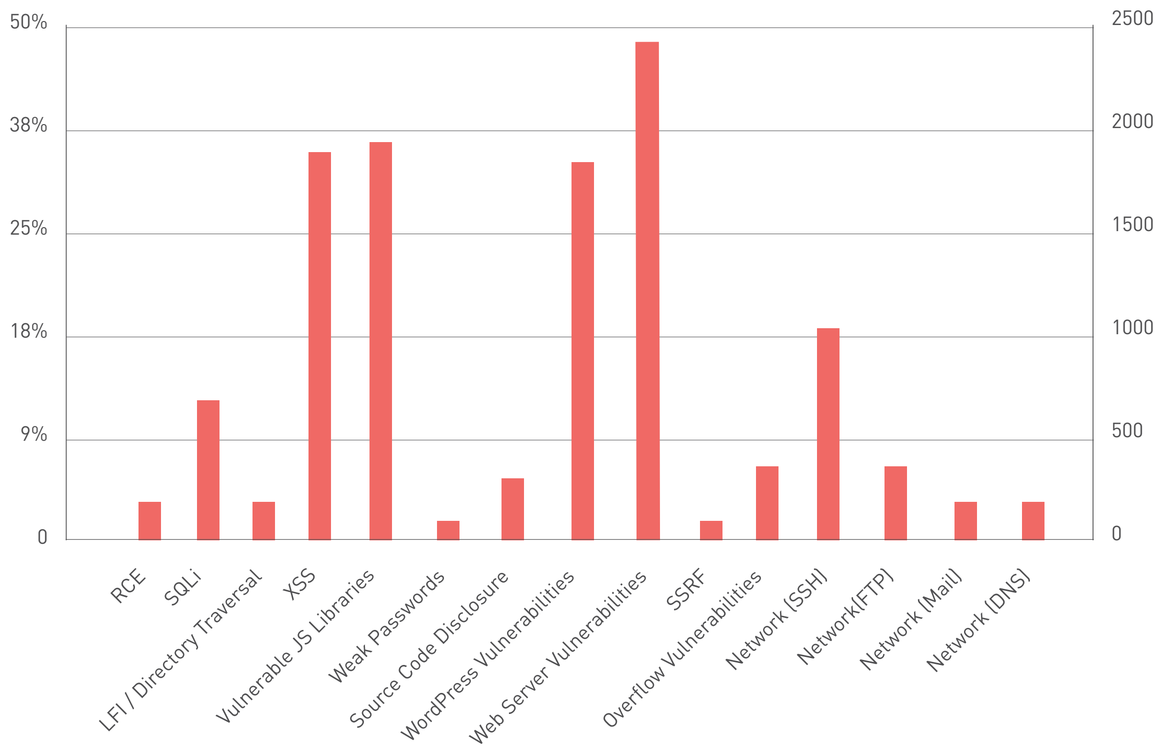


Figure 2.

High Vulnerabilities Analysis.

Vulnerabilities at a Glance

Medium Severity

This chart lists the vulnerabilities types found that fall into our Medium Severity Category.

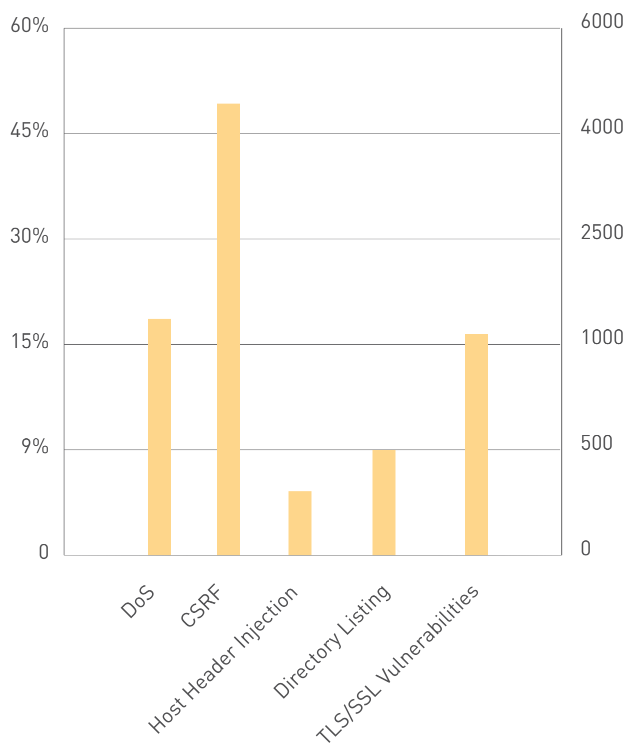


Figure 3.

Medium Vulnerabilities Analysis.

We utilize Acunetix to more thoroughly assess internet-facing websites and servers. Acunetix helps us identify vulnerabilities in conjunction with other vulnerability scanning applications. Acunetix has been a more reliable application when discovering / determining different types of malicious code injection vulnerabilities (SQL, HTML, CGI, etc).

Carter Horton, Assoc. Information Analyst, **GD Information Technology**

Vulnerability Severity

Vulnerabilities are weaknesses in a system, such as a web application, network perimeter device, or an appliance. Vulnerabilities create security threats – negative events that can lead to an undesired outcome, as well as unknown and potentially more significant damage.

Not all vulnerabilities threaten an organization in the same way, because different vulnerabilities have different impacts given where and how they're exploited. While it's essentially impossible to provide a metric that will work for every organization in all situations, Acunetix uses vulnerability severity as a way of classifying the potential impact a given security vulnerability poses. In turn, organizations can use additional vulnerability management features such as Business Criticality within Acunetix to better map out the degree to which a vulnerability may become a business risk.

The Severity Level of a vulnerability is therefore assigned based on the impact of that vulnerability if successfully exploited, together with the degree of difficulty involved in exploiting it. The result of a successful attack could vary from information disclosure to a complete compromise of applications or systems.

The following gradation provides a description of the impact of each vulnerability Severity Level in the results of this analysis.

High Severity	Medium Severity	Low Severity
This level indicates that an attacker can fully compromise the confidentiality, integrity or availability of a target system without specialized access, user interaction or circumstances that are beyond the attacker's control. It is very likely to allow lateral movement and escalation of the attack to other systems on the internal network of the vulnerable application.	This level indicates that an attacker can partially compromise the confidentiality, integrity or availability of a target system. Specialized access, user interaction, or circumstances that are beyond the attacker's control may be required for an attack to succeed. It is very likely to be used in conjunction with other vulnerabilities to escalate an attack.	This level indicates that an attacker can compromise the confidentiality, integrity or availability of a target system in a limited way. Specialized access, user interaction, or circumstances that are beyond the attacker's control is required for an attack to succeed. It needs to be used in conjunction with other vulnerabilities in order to escalate an attack.

Vulnerability Analysis

Remote Code Execution •

Remote Code Execution (RCE) is usually the worst case scenario in a web application attack, since it allows an attacker to execute arbitrary code within a web application.

This can be escalated to running operating system commands. An attacker may gain persistence by using a reverse shell. A reverse shell is a commonly used method for an attacker to gain interactive control of the victim machine by having it initiate an outbound connection to the attacker.

The attacking machine would have a listener port on which it receives the connection. This method is frequently used due to its ease as a method for bypassing firewall restrictions, since – unlike inbound connections – outbound connections are typically allowed, or at least lax.

Once an attacker gains control of a system through RCE, the potential is there for the attacker to take over the system entirely. If an attacker gains access to a single system, they can attempt to achieve lateral movement by compromising connected systems.

They can achieve this by taking note of resources on the internal network and seeking opportunities for collecting additional credentials and abusing local privilege escalation vulnerabilities.

Analysis

Two percent of sampled targets were found to be vulnerable to RCE. While it is fortunate that this vulnerability is not more prevalent, given its potential impact, this is still an alarming figure.

While there are some cases where the execution of arbitrary code or, more commonly, operating system commands, is required, such cases should be remediated immediately, and follow up steps taken to ensure they are less likely to occur again.

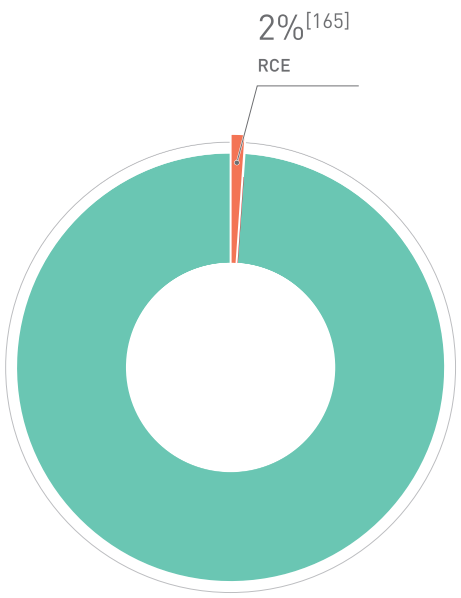


Figure 4.

RCE Analysis.

SQL Injection (SQLi)

SQL injection (SQLi) is a type of attack in which a hacker can take advantage of the insecure SQL query a web application makes to a database server (such as MySQL, Microsoft SQL Server and Oracle). It exploits weaknesses in a web application that are usually the result of poor development practices or mistakes. Using an SQL injection, an attacker can send SQL commands to the database server, allowing them to gain unauthorized access to data or, in extreme cases, even take over the entire system on which the database server is running.

SQL injections are among the oldest, most prevalent and dangerous web application vulnerabilities. Since SQL injections affect web applications that make use of an SQL database, virtually every type of web application needs to pay attention to it. SQLi also happens to be one of the most well-understood web application vulnerabilities, with hundreds of free, ready-made tools to make it quicker and easier for attackers to take advantage of SQL injection vulnerabilities.

By abusing an SQL injection vulnerability, an attacker may be able to bypass a web application's authentication and authorization mechanisms, retrieve the contents of an entire database, and even add, modify and delete records in that database, impacting its data integrity.

In this worst-case scenario, SQL injections can provide an attacker with unparalleled access to sensitive data, such as customer data, Personally Identifiable Information (PII) and other sensitive information.

While SQLi is usually used by attackers to steal data from databases, such vulnerabilities may be escalated to gain even further access, especially if the database server is not correctly configured or is configured insecurely. For instance, an attacker may take advantage of an SQL injection vulnerability and use it to delete valuable records, or even entire tables from a database, effectively causing a Denial of Service attack. In other cases, an attacker may use the SQL injection vulnerability to write files in the system. This can potentially lead to an attacker uploading a web shell onto the server and subsequently taking over the entire server, or pivoting into other systems.

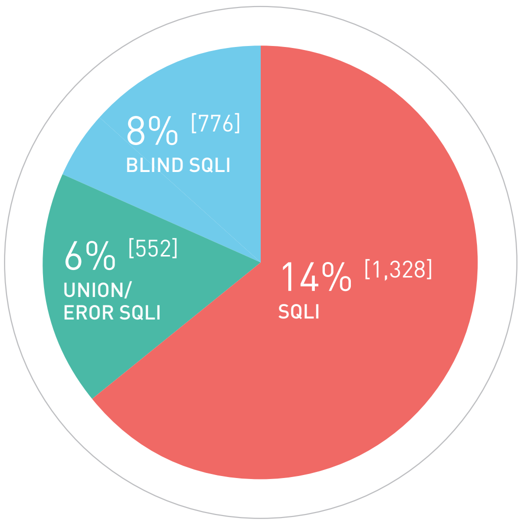


Figure 5.

SQLi Analysis.

Blind SQL Injection •

Blind SQL Injection is a variation of SQLi which is usually used by an attacker as a last resort when faster ways to exploit the SQL injection vulnerability are not possible.

This does not mean that SQLi would not be possible. However, the attack does take significantly longer, and in some cases it may be easier to detect due to the sheer volume of requests an attacker needs to send.

The reason for this is that the attack is 'blind'. In such an attack, an attacker can not simply display data within the response received from the web application. Instead, the attacker must use a side channel to retrieve the data they want. For example, an attacker may be able to retrieve data from the database through a Blind SQLi vulnerability, by sending crafted queries to the database server containing logical statements, and asking the database server to wait a specified amount of time if a condition is true.

Analysis

Fourteen percent of sampled targets were vulnerable to at least one SQL injection. This figure has been slowly decreasing: it went from 26% (2015) to 14% (2018), indicating that developers are becoming wiser to SQLi and its perils. With support for parameterized SQL queries in nearly all major programming languages and frameworks, SQL injection should be a very well understood problem and easy to prevent. The reality, however, is that due to a cocktail of legacy applications, developers may not yet be fully aware of the dangers of SQLi. Human error, mixed with old habits, is difficult to overcome.

Unsurprisingly, Blind SQLi (8%) was found to be a more prevalent SQL injection vector than others (UNION and error-based SQL injection). This is primarily due to the fact that many application developers suppress database error warnings.

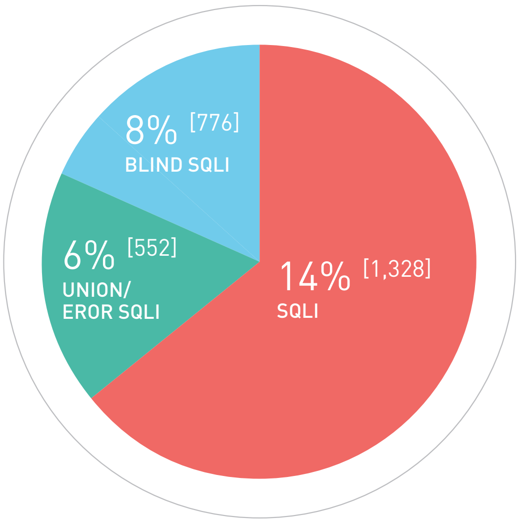


Figure 5.

SQLi Analysis.

Local File Inclusion & Directory Traversal •

Local File Inclusion (LFI) and Directory Traversal are similar vulnerabilities with very different outcomes.

Directory Traversal or Path Traversal attacks manipulate web application inputs by using the dot-dot-slash (../) sequences, or similar variations. The ../ (or sometimes ..\ in Windows systems) is a convention indicating that you go up (traverse) a directory (folder). This means that if a web application is designed to read a file from the filesystem, and the attacker has control over the filename, they can simply replace it with something similar to ../../../etc/passwd and gain access to that system's files.
An attacker can not only read any file to which the web server has access, but on Linux systems, an attacker can also read data from the /proc virtual filesystem. This can be used by the attacker to gain information about the operating system's kernel version, mounted file systems, routing table and other attributes that can lead to information disclosure.

While it is possible for an attacker to use an LFI vulnerability in a similar way to a Directory Traversal vulnerability, unlike Directory Traversal (which only allows an attacker to read files), LFI is a type of inclusion vulnerability. This means that it includes files and executes them as code.

Developers, especially in languages such as PHP and JSP, make frequent use of 'includes' to avoid repeating code across multiple pages. If an attacker can combine a Local File Inclusion vulnerability with a malicious file upload vulnerability, then the attacker can escalate the LFI to a Remote Code Execution (RCE).

Analysis

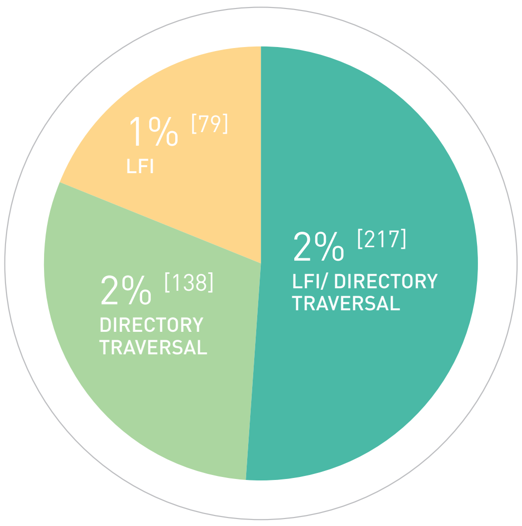
Two percent of sampled targets were vulnerable to Directory Traversal, while a further 1% of targets were found to be vulnerable to Local File Inclusion.

While it is not always straightforward for an attacker to escalate an LFI vulnerability up to a Remote Code Execution attack, the vulnerability is certainly not something to take lightly.

Also, Directory Traversal attacks may provide an attacker with crucial information about the internal workings of an application, as well as the host system it is running on, making it a serious information disclosure concern.

Figure 6.

LFI and Directory Traversal Analysis.



Cross-site Scripting (XSS)

Cross-site Scripting (XSS), unlike most vulnerabilities that affect server-side resources, is a vulnerability that arises on the client side. Cross-site Scripting can be thought of as a client-side code injection that occurs, predominantly, through the use of JavaScript.

Cross-site Scripting vulnerabilities can be classified in the following ways:

- 🟥 Stored (Persistent) XSS
- 🟦 Reflected (Persistent) XSS
- 🟧 DOM-based XSS

While there are a number of variations of XSS, all cases follow a similar pattern. The attacker's objective is to make a victim inadvertently execute a maliciously injected script, that runs in the context of a trusted web application. The purpose of this is to steal sensitive data to which the user has access, or even to modify the layout of the web application so that the user is encouraged to submit sensitive data to the attacker. An attacker's malicious script is often referred to as a (malicious) payload.

Stored (Persistent) XSS attacks allow an attacker to inject a payload that is permanently stored (persisted) by the target application and 'echoed' back when a victim visits a page. Stored XSS is the most dangerous type of XSS because the attacker only needs to make a single request, so it can affect any victim who visits the page. One of the most famous examples of a stored XSS attack is the Samy Worm, a (thankfully harmless) worm that spread through MySpace back in 2005.

Reflected XSS attacks allow an attacker to trick a victim into following a link containing an XSS payload (usually achieved through phishing or other types of social engineering). Once the payload is sent to a web page vulnerable to reflected XSS, the payload is included back (reflected) as part of the HTTP response from the server.

DOM-based XSS is an advanced type of XSS that allows a skilled attacker to craft a payload. This payload is executed as a result of legitimate JavaScript modifying the Document Object Model (DOM) in a victim's browser. In contrast to the other types of XSS, with DOM-based XSS the payload is often not present within the HTTP response. Client-side code designed to process elements in the DOM (referred to as a DOM-XSS sink) executes the malicious payload that has been injected in a DOM element that the attacker has control over (referred to as a DOM-XSS source).

When a web application is vulnerable to XSS, it executes the attacker-supplied content from a source that the application implicitly trusts, rather than treating that input as untrusted and encoding or filtering it appropriately.
In each case, XSS results in the browser interpreting the attacker's payload as legitimate JavaScript code and subsequently executing it.

The consequences of Cross-site Scripting are not immediately obvious. This is especially true since modern web browser vendors take security seriously, and take several precautions when running JavaScript within a sandboxed environment. However, if you consider that an attacker's JavaScript payload has access to all the same objects as the rest of the web page, including access to cookies (often used to store session tokens), an attacker may easily impersonate users, steal sensitive data or trick users into divulging sensitive information.

Browser vendors have made efforts to filter reflected XSS in order to make it harder for attackers to carry out reflected XSS attacks. But attackers are good at evading filters. Additionally, modern browsers now incorporate Content Security Policy (CSP) in an effort to make XSS harder for an attacker to abuse. Unfortunately, CSP's adoption is still very low, and it may be incredibly tricky for some sites to implement properly.

Analysis

A staggering 32% of sampled targets were vulnerable to one form or another of Cross-Site Scripting. XSS, combined with social engineering, makes it possible for attackers to steal cookies and impersonate users, and engage in keylogging, phishing and identity theft.

Critically, XSS vulnerabilities provide attackers just what they need to convert attacks to more serious ones.

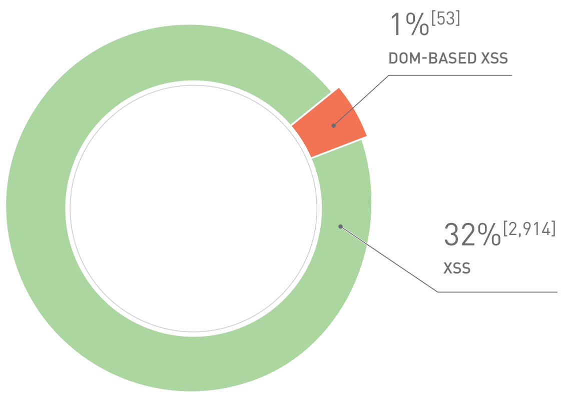


Figure 7.

XSS Analysis.

Vulnerable JavaScript Libraries

Vulnerable JavaScript libraries, like many other software supply chain tools and technologies, were created to make development faster and easier. Several web applications rely on old or outdated JavaScript libraries (i.e. old and vulnerable versions of jQuery) opening the door to Cross-site Scripting vulnerabilities.

Analysis

A phenomenal 33% of sampled targets were found to use JavaScript libraries with known XSS vulnerabilities. The most frequently encountered vulnerable JavaScript libraries were old versions of jQuery and jQuery UI, followed by old versions of Moment.JS and the YUI Library.

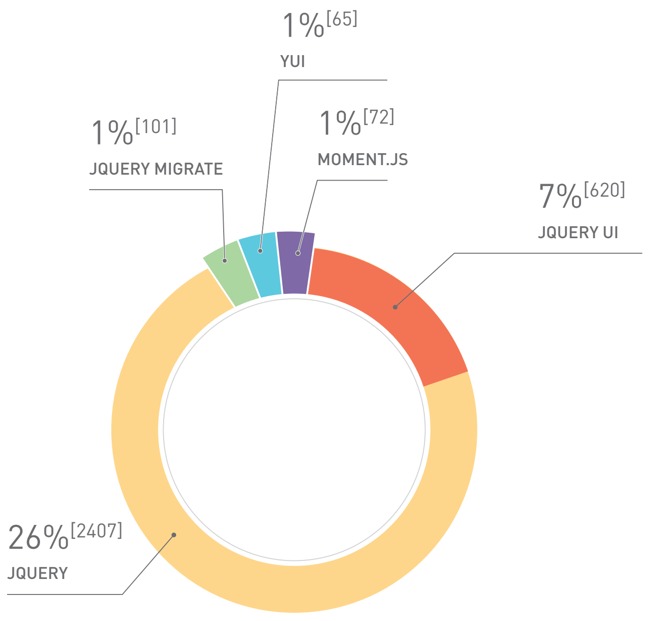


Figure 8.

Vulnerabilities JavaScript Analysis.

Weak Passwords •

Weak passwords are usually short, commonly-used words or defaults that can be easily guessed by an attacker when they encounter a login prompt. In some cases, weak passwords may be guessed by running a dictionary attack, using common dictionary words and common passwords, names, words based on the username or variations on these themes. In many other cases, weak passwords are simply default username and password combinations like admin/admin or admin/password.

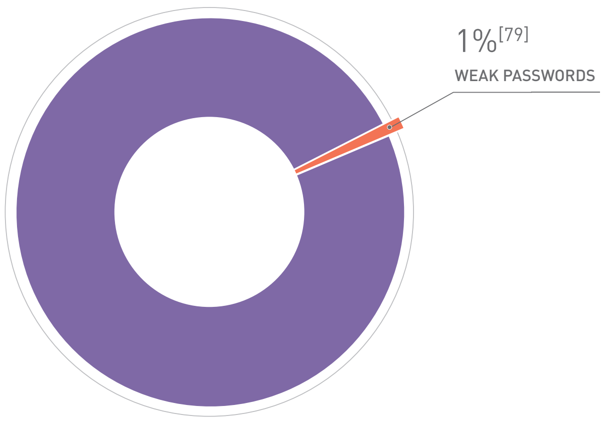
Weak passwords are the chink in the armor of even a well-defended system. They allow an attacker to easily gain access to administrative portals, where they can make configuration changes, upload files or perform other actions in order to escalate their attack.

Analysis

One percent of sampled targets were found to use weak or default passwords. Considering that this is both a trivial matter to resolve, as well as potentially very dangerous (depending on the system to which an attacker may gain access), it's good to discover that this vulnerability is not more prevalent.

Figure 9.

Weak Passwords Analysis.



Source Code Disclosure •

Source code is best kept away from an attacker's prying eyes, especially if the software running is not open source. If it is open source, an attacker already has access to it. If it isn't open source, it may contain clues about other vulnerabilities an attacker could exploit to gain access to a system.

Even if the software run is open source, many software packages configure secrets and other important parameters within source files which are then uploaded to the server and used by the respective application during runtime. But regardless of the source code's licensing status, it's best that potential attackers are simply unable to explore your website or web application's source code.

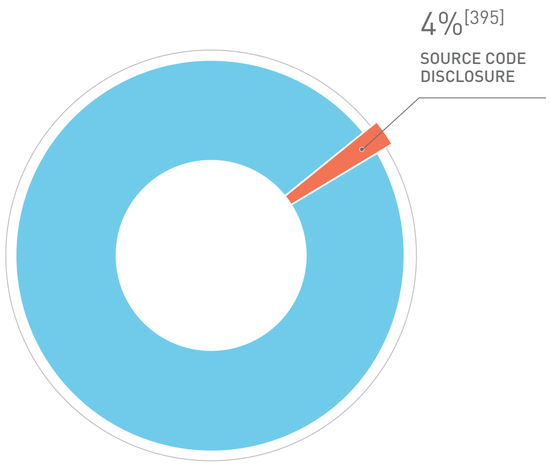
Source code and related configuration files may divulge sensitive configuration information about database credentials, or information on how your web application functions. An attacker could employ this information to escalate an attack by exploiting other vulnerabilities or misconfigurations discovered through the disclosed source code or configuration files.

Analysis

Four percent of sampled targets were found to contain one or more instances of Source Code Disclosure. This vulnerability could provide an attacker with crucial information about the inner workings of an application. It could be an opportunity for them to download and analyze the source code for additional vulnerabilities, comments or even hints about the underlying infrastructure on which the application runs.

Figure 10.

Source Code Disclosure Analysis.



Server-side Request Forgery •

Server Side Request Forgery (SSRF) allows an attacker to create (forge) requests from a vulnerable server. While this may not seem immediately useful, SSRF becomes very dangerous when an attacker can forge requests to internal systems, or even services running on localhost.

SSRF is useful to an attacker who wants to bypass firewall controls. By sending and receiving responses to requests made to systems that sit on internal networks behind firewalls, an attacker may be able to access and further exploit hosts and services not accessible from the outside world. Internally deployed systems that are particularly vulnerable to SSRF are those which, by default, do not require authentication, or which provide some results without authentication. Examples of such services are databases (Elasticsearch, MongoDB), caching (Memcached, Redis), service discovery (Consul, Etcd), and monitoring systems (Prometheus, Graphite).

SSRF may also be used to list and even exploit other internal web applications containing vulnerabilities of their own.

Figure 1.

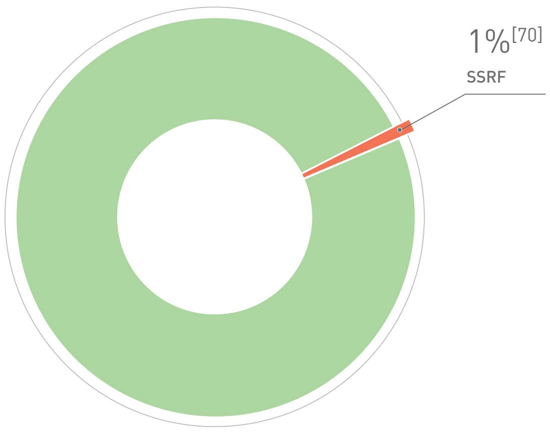
Vulnerabilities Analysis.

Analysis

One percent of sampled targets were found to be vulnerable to Server Side Request Forgery. While not as widespread as other high severity vulnerabilities such as SQL injection or Remote Code Execution, SSRF can have major potential impacts. Once an attacker can make requests to an internal network, they may easily access services which lack authentication, launch Denial of Service attacks against internal services, or conduct in-depth reconnaissance on an internal network.

Figure 11.

SSRF Analysis.



Overflow Vulnerabilities •

Overflow vulnerabilities, such as buffer overflow, stack overflow, and heap overflow, exist when a program does not exercise proper bounds checking. As a result of this, user input may overflow the buffer's boundary and overwrite adjacent memory locations, while writing data to the buffer.

Overflow vulnerabilities will only occur in programs written in languages that are not memory safe (C and C++). Overflow vulnerabilities can corrupt data, crash programs, or, in many cases, even allow the execution of malicious code. This makes overflow vulnerabilities incredibly destructive in software used to run web applications or network infrastructure, such as web servers, routers, and mail servers.

Analysis

Five percent of sampled targets were found to be vulnerable to types of Overflow vulnerabilities, like buffer overflows, integer overflows, heap overflows and stack overflows.

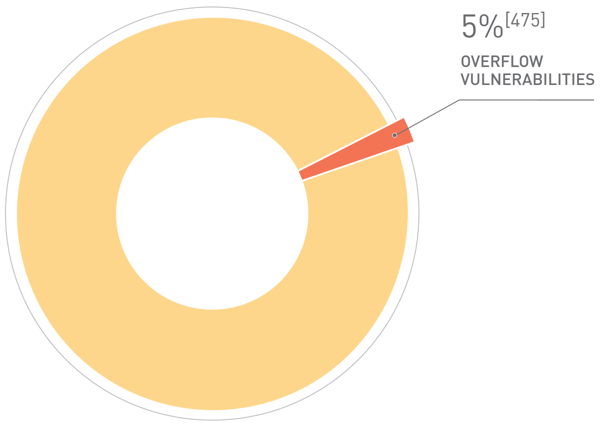


Figure 12.

Overflow Vunerabilities Analysis.

Perimeter Network Vulnerabilities

Resources residing on the network perimeter are usually devices or services designed to be internet facing. Examples of these are router's VPNs and firewalls, web servers and load balancers, and even DNS and mail servers. In most cases, vulnerabilities within such services can be traced to old versions of some critical piece of software; in other cases, misconfigurations are to blame.

Since many organizations have already moved a significant portion, if not all, of their infrastructure to one or more public clouds, the perimeter is no longer as clear-cut as it used to be. Instead, the perimeter is constantly changing to include everything from corporate on-premise firewalls to web server infrastructure in the cloud.

A vulnerability or misconfiguration in a critical service may result in serious information disclosure or even bypass authentication controls, allowing an attacker to escalate an attack.

Analysis

Nineteen percent of sampled targets were found to be vulnerable to SSH-related vulnerabilities.

Six percent of sampled targets were found to be vulnerable to FTP-related vulnerabilities, the majority of these vulnerabilities being low-severity vulnerabilities or misconfigurations. Most of the issues identified revolve around information and version disclosure of FTP servers.

Two percent of sampled targets were found to be vulnerable to Mail-related vulnerabilities.

Two percent of sampled targets were found to be vulnerable to DNS-related vulnerabilities.

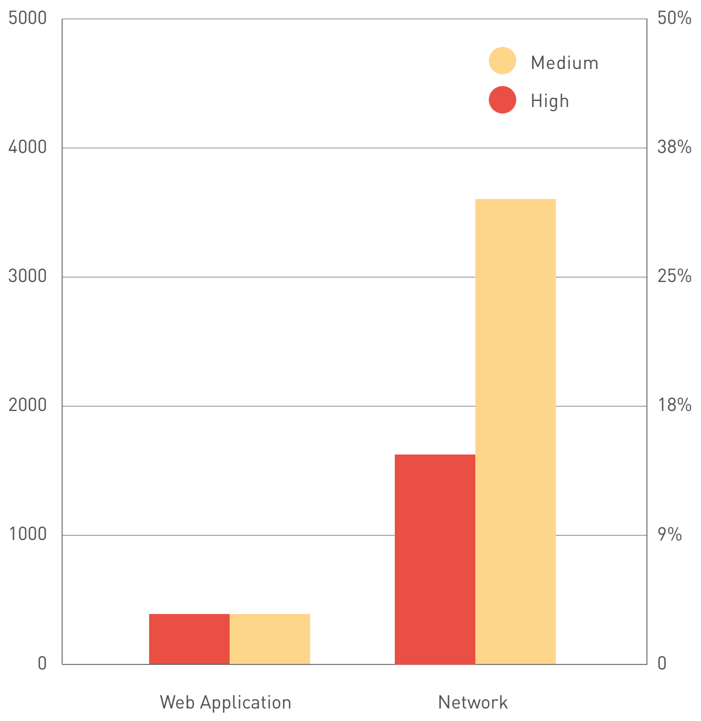


Figure 13.

Perimeter Network Analysis.

DoS Related Vulnerabilities

Denial of Service (DoS) is a category of attack in which an attacker aims to render a system non-responsive for a period of time, usually by overwhelming or 'flooding' the target with traffic, so that legitimate traffic is prevented or interrupted.

Traditionally, DoS attacks focus on exhausting resources of network infrastructure devices or services such as routers, load balancers and web servers in a volumetric Denial of Service attack. While volumetric attacks are increasingly common on the internet, attackers can also take advantage of a less common, but potentially more damaging, type of DoS attack – Application-based Denial of Service.

Since DoS attacks make servers, services or network infrastructure unavailable to its intended users, depending on the impacted systems, it may result in a loss of business as well as increased costs associated with additional resource use in addition to the extra data transfer costs required to stay online. DoS vulnerabilities are also among the hardest to defend against, as there is often no foolproof way of stopping them.

In an Application-based Denial of Service incident, an attacker focuses on sending fewer requests (less expensive for the attacker) which cause more damage to the target (more expensive for the victim). An example of such a request is if an attacker discovered that an API endpoint of a web application had to complete a lot of processing before returning a result.

They could conclude that such a request returned a lot of data and took a long time to process a result, and then send requests to that specific endpoint to bring the web application down more quickly.

Analysis

Eighteen percent of sampled targets were found to be vulnerable to Denial of Service vulnerabilities, while 13% were vulnerable to a specific kind of application-level DoS known as Slow HTTP Denial of Service (also known as Slowloris).

Slowloris is an attack that allows an attacker to use up all of a web server's connections with minimal bandwidth. The attacker achieves this by sending HTTP requests to a web server without closing the connection. This causes the web server to run out of the maximum HTTP open connections allowed. As a result, the server drops requests made by legitimate users until the attack ends.

Given the business impact associated with DoS attacks, they should be taken seriously. But mitigating DoS attacks is difficult and no one is immune to them, especially if an attacker has sufficient resources. However, there is much that can be done to mitigate application-level DoS vulnerabilities like the Slow HTTP DoS attacks, to make it harder for attackers to achieve DoS through the use of simple vulnerabilities.

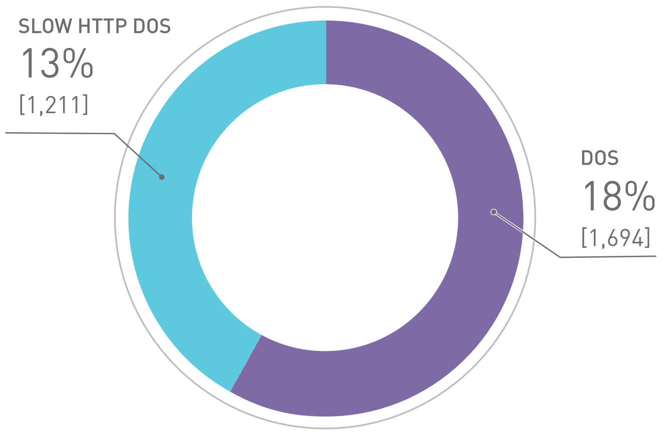


Figure 14.

DoS Related Analysis.

Cross-site Request Forgery •

Cross-Site Request Forgery (CSRF) vulnerabilities allow attackers to trick victims into making an authenticated HTTP request to a vulnerable website or web application from another origin (often, but not always, from a website the attacker controls).

By abusing a CSRF vulnerability, an attacker takes advantage of the trust a website has with a victim's browser. Whenever a user sends an HTTP request to a website or web application, the browser will automatically send with it any cookies associated with that website's origin (the domain from which the the HTTP request originates). As a result, if an attacker convinces a victim to visit a page that sends an HTTP request, the browser will automatically, and crucially – without user intervention – include a valid cookie as part of the request, provided one was set. Cookies contain authentication tokens to enable users to log in, so, if a victim is logged into a vulnerable website, the HTTP request would be made in the context of the logged in user.

CSRF attacks are potentially dangerous because they leverage the identity and privileges of the victim as part of the forged request (assuming the victim is logged in).

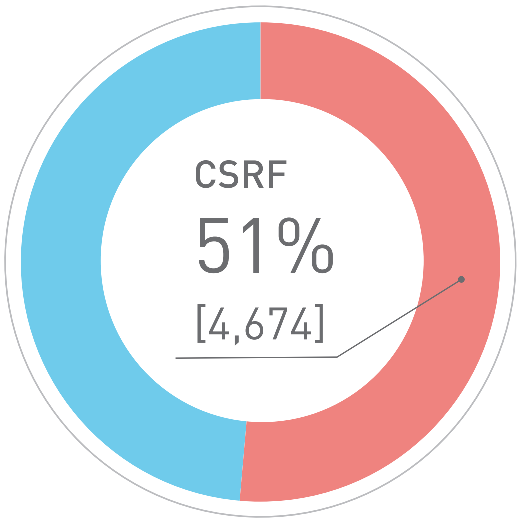
Analysis

Fifty-one percent of sampled targets were found to be susceptible to Cross-site Request Forgery, or have an HTML form without the presence of a CSRF token.

While it is possible to detect CSRF automatically, it is not possible to automatically determine its real-world impact. The reason for this is because not every HTML form necessarily has sensitive actions associated with it (for example, a search query).

Figure 15.

CSRF Analysis.



Host Header Injection •

Host header injection is an attack that occurs due to the implicit trust of web applications in the HTTP host header. Web application developers or technical operations engineers may trust the value of the HTTP host header within their applications, web servers, or load balancer configurations. However, this trust in the HTTP host header is misplaced, since an attacker has control over this header when crafting a malicious HTTP request.

Some applications implicitly trust the value inside the HTTP host header and use it for everything from generating links to importing scripts and stylesheets on a page. Application developers can even rely on the value of the HTTP host header to generate password reset links.

Under the right circumstances, an attacker can use control of the HTTP host header to cause web cache poisoning. This tactic allows an attacker to temporarily 'poison' a web cache, resulting in that cache serving victims whatever malicious content the attacker wants. Where the HTTP host header is used to reset password links, an attacker can send a reset password link to a different email address instead of the one to which the email was intended. This results in an attacker possessing the ability to take over an account entirely (especially if no two-factor authentication is in place).

Analysis

Four percent of sampled targets were found to be vulnerable to Host Header Injection. While Host Header Injection does bear a significant risk, it isn't the most straightforward to exploit. It requires unlikely conditions to be met for an attacker to successfully take advantage of it. Host Header Injection like XSS and SQLi is another case of placing implicit trust in user-controlled input, which could be easily avoided.

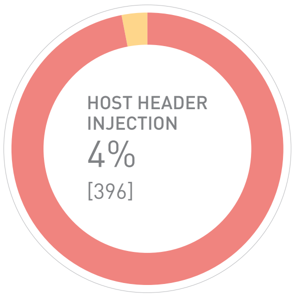


Figure 16.

Host Header Injection Analysis.

Directory Listing •

Directory Listing is a web server "feature" which could result in a web server divulging sensitive information to an attacker, if it is not disabled. Directory Listing is enabled by default in web servers like Apache HTTP Server. This allows a user to view a list of files and directories hosted on the web server in an organized, hierarchical way. An attacker can abuse this feature by navigating towards a directory without an index file. If the directory listing is enabled, the web server will provide the attacker with a list of files and directories.

Directory Listing may allow an attacker to escalate an attack by disclosing sensitive information or even configurations. For instance, an attacker may leverage Directory Listing to download source code configuration files, as well as private information.

Analysis

Nine percent of sampled Targets were found to be vulnerable to Directory Listing misconfigurations. While this result is not surprising, given that directory listing is enabled by default on Apache HTTP Server, system administrators should follow basic hardening guides when deploying services with which they may not be fully familiar.

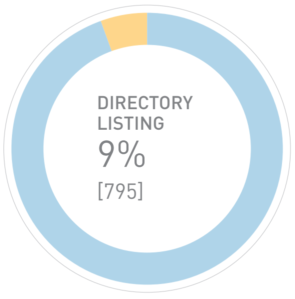


Figure 17.

Directory Listing Analysis.

TLS/SSL Vulnerabilities

Transport Layer Security (TLS) and its predecessor, Secure Socket Layer (SSL), are used every day to authenticate, encrypt and verify the integrity of data between a client and a server.

TLS is of monumental importance to every website on the Internet, and even more so to websites that deal with sending and receiving sensitive data, such as login information, personal information and credit cards. TLS is used to ensure that no third-parties can access or alter these communications while data is in transit. TLS misconfigurations, or using old or broken TLS ciphers that allow downgrade attacks, is an issue that may impact the privacy, as well as the integrity, of data passing over a public network, such as the Internet.

Analysis

Seventeen percent of sampled targets were found to have TLS/SSL issues. Hyped TLS vulnerabilities like Heartbleed (0%) and POODLE (3%), which were very much widespread when they were initially discovered, have now decreased significantly, with Heartbleed almost entirely absent.

DROWN, which is a TLS downgrade attack, affects 1% of Targets, while BREACH, which is TLS a vulnerability allowing the extraction of repeated secret tokens, affects 6% of sampled targets.

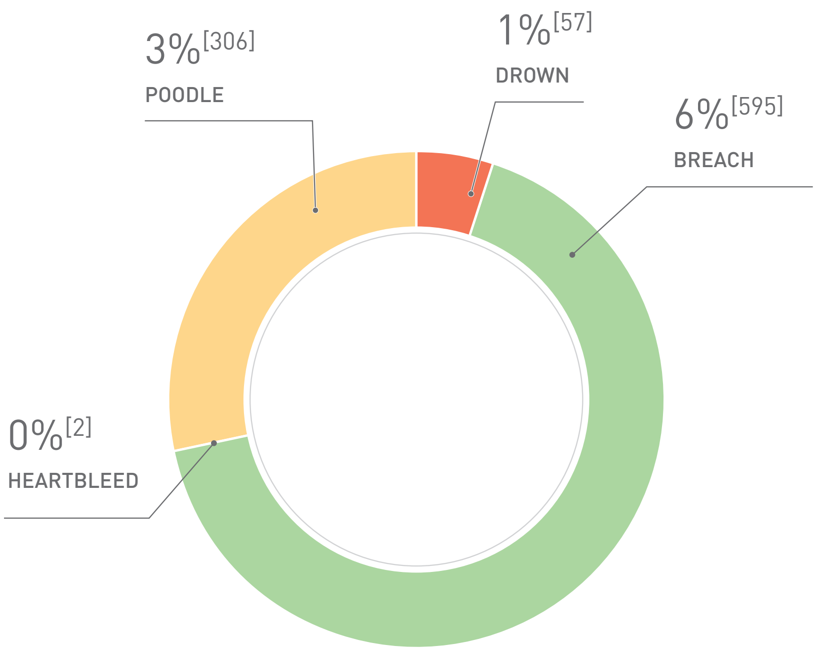


Figure 18.

TLS/SSL Vulnerabilities Analysis.

WordPress Vulnerabilities

It is estimated that over 30% of all websites on the Internet are based on WordPress.

With such a massive number of installations deployed around the world, the popularity of Wordpress makes it a prime target for attackers. While WordPress does expose some inherent security weaknesses, such as username enumeration and XML-RPC authentication bruteforcing, the WordPress community works hard to make WordPress secure by default.

Thanks to the strong community behind WordPress, any security vulnerabilities that affect the WordPress core are easy to fix. Users just need to make sure they're running the latest version of WordPress, in which security updates are now turned on automatically, making this a zero effort endeavor. Unfortunately for the estimated 32% of sites on the Internet running out of date versions of WordPress, upgrading may involve more effort due to incompatible old plugins and themes.

These positive points about WordPress' core security efforts cannot be applied to its plugin and theme ecosystem. Plugins and themes provide WordPress site authors a way to extend WordPress' core functionality. As an open system, it's possible for anyone to write a plugin and distribute that plugin via the WordPress plugin repository. Consequentially, it is very common for popular plugins to contain critical vulnerabilities that make their way onto thousands of WordPress installations.

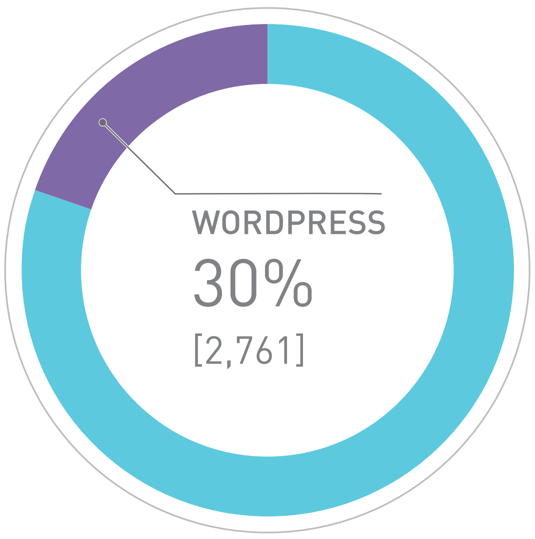
Vulnerabilities within popular plugins can range from information disclosure to Cross-site Scripting, all the way to the most perilous of vulnerabilities, such as SQL injection and Remote Code Execution. Once an attacker finds a vulnerability in a popular WordPress plugin, their tendency is to exploit the vulnerability across thousands of WordPress sites across the Internet.

Analysis

Thirty percent of sampled targets were found to be vulnerable to one or more WordPress vulnerabilities. The impact of a vulnerable WordPress core installation or that of a vulnerable WordPress plugin will vary, based on the kind of vulnerability in question. This may range from Cross-site Scripting to SQL injection, all the way to Remote Code Execution.

Figure 19.

Wordpress Analysis.



Web Server Vulnerabilities & Misconfigurations

Web server software, like all other software, has sporadic security issues. Usually, web server vendors are quick to patch any critical vulnerabilities. Upgrading to the latest version of the web server's software is sufficient to resolve the security issue.

Failure to patch even trivial vulnerabilities in web server software can lead to significant compromise. It depends on the level of the vulnerability. But by their nature, web servers are designed to be public facing and are meant to deal with all sorts of traffic on the Internet.

Vulnerabilities in web servers may range anywhere from information disclosure all the way to a remotely exploitable buffer overflow vulnerability, which could allow an attacker to escalate an attack to Remote Code Execution (RCE).

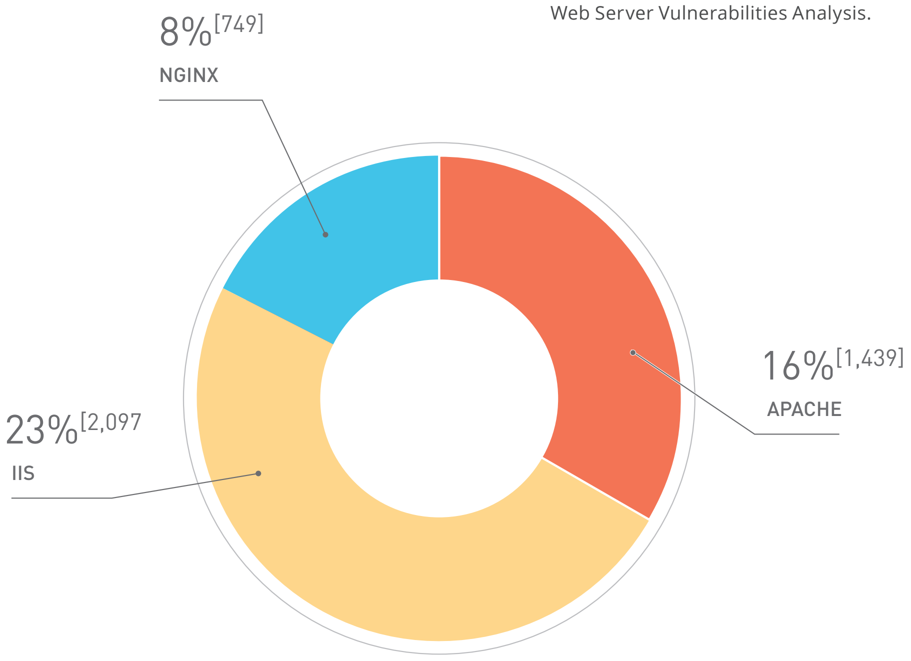
Analysis

Forty-six percent of the sampled targets were found to have either Web Server Vulnerabilities or Misconfigurations. Unsurprisingly, the large majority of misconfigurations in this category were related to version disclosure. It's common for various web servers to disclose not only which web server is serving a request, but also what version. While this is not strictly classified as a vulnerability, it may provide an attacker with some useful, albeit limited, information.

In other cases, old versions of web servers were identified that contained vulnerabilities, mostly related to Denial of Service or Information disclosure.

Figure 20.

Web Server Vulnerabilities Analysis.



Conclusion

The analysis of this year's results continues to reaffirm that the web application vector is a major and significant threat to the security of organizations of any size, anywhere in the world, whether they take security steps or not.

The traditional patching approach to mitigating most network-layer vulnerabilities just doesn't work the same with web application vulnerabilities, such as SQL injection, Cross-site Scripting, and Remote Code Execution. This is due to the fact that web application vulnerabilities tend to occur due to poor design choices made during the development and deployment phases.

There is overall progress in the number of high severity vulnerabilities detected. But over 35% of websites and web applications are susceptible to at least one single high severity vulnerability. More needs to be done to keep up with the speed at which these vulnerabilities are introduced. At Acunetix, we believe that in order for many organizations and their security teams to have a fighting chance at keeping up, security needs to be automated and incorporated as an integral part of the development process.

Over the past couple of years, Acunetix – through both Acunetix Online, as well as Acunetix On-Premise – has worked hard to make web security accessible to as many stakeholders as possible within an organization. Making Acunetix accessible through a powerful, easy to use, multi-user web interface, while at the same time keeping all the important technical information, controls and insights at the fingertips of the security practitioners, has been at the forefront of our Product Team's mission.

By making Acunetix up to 50% faster, and providing even more integration options than ever before, it's now easier than ever to embed Acunetix at the heart of your development and Continuous Integration (CI) processes. You no longer have to waste time weeding through an endless stream of false positives, or waiting an excessive amount of time for a scan to finish.

Web application vulnerabilities pose an increasingly serious threat to the security posture of organizations as a whole. Now is the best time to take a serious look at how automation can help make your application security more holistic and agile.

About Acunetix Online

With an easy to use user interface, dead-accurate scan results, and unparalleled technology support, Acunetix Online makes scanning for web application vulnerabilities easy and painless, whether you manage a handful or hundreds of websites.

With support for the latest and best technologies, including HTML5 and JavaScript, Acunetix Online detects a wide range of dangerous vulnerabilities, including SQL Injection, Cross-site Scripting, and Local File Inclusion. It does all of this while lowering false positives to an industry-wide minimum, and keeping a relentless focus on scan speed without sacrificing accuracy.

With over 3100 web application vulnerability tests, Acunetix Online provides the widest testing coverage available, including the detection of out-of-band vulnerabilities such as Blind Cross-Site Scripting (BXSS), Server-Side Request Forgery (SSRF), XML External Entity Injection (XXE), Host Header Injection and more.

Acunetix Online incorporates with the most advanced detection of WordPress, Drupal, and Joomla! vulnerabilities, as well as a wide range of reports including OWASP Top 10, PCI DSS and HIPAA compliance.

Give your application security programme the boost it deserves. Get a demo of Acunetix today.

www.acunetix.com/web-vulnerability-scanner/demo/

About Acunetix

Founded in 2005 to combat the growing web application security threat, Acunetix is a technology leader and pioneer in the realm of automated web application security.

Acunetix is depended on globally and used by individual pen-testers and security practitioners, all the way to Fortune 500 and Government organizations such as NASA, the U.S. Air Force, Disney and Adobe.

acunetix

WHERE TO FIND US

Stay up to date with the latest web security news.

Website. www.acunetix.com

Acunetix Web Security Blog.
www.acunetix.com/blog

Facebook. www.facebook.com/acunetix

Twitter. twitter.com/acunetix

CONTACT INFORMATION

Acunetix (Europe and ROW)
Tel. +44 (0) 330 202 0190
Fax. +44 (0) 30 202 0191
Email. sales@acunetix.com

Acunetix (USA)
Tel. (+1) 737 241 8773
Fax. (+1) 737 600 8810
Email. salesusa@acunetix.com